»Лекционен курс

»ООП1 (Java)

Композиция ›

Многократно използване на имплементацията

Въведение

- Една от най-съществените характеристики на Java е многократното използване
  - Предоставя значително повече възможности от копиране и промяна на кода
- Решението на този проблем (както всичко друго в Java) е свързано с класовете
  - Многократното използване на код като създаване на нови класове
  - Вместо да ги създаваме от самото начало можем да използваме съществуващи класове
  - Грешките в тях са вече отстранени
- Цел: използване на класове без добавяне излишни неща в съществуващия код

Многократно използване на имплементацията

- Многократното използване на код е едно от най-големите предимства на ООП
- Най-прост начин за многократно използване
  - Създаване и използване на обект от един клас
  - Създаване на член-обект - поставяне на обект от съществуващ в нов клас

Многократно използване на имплементацията

- Композиция: новият клас може да бъде съставен от произволен брой и тип други обекти в произволна комбинация, необходима за постигане на желаната функционалност
  - Притежава голяма гъвкавост
  - Член-обектите обикновено са недостъпни за клиентите (private)
    - Така можем да променяме тези обекти без да влияем върху кода на клиентите
  - Освен това, можем да променяме член-обектите по време на изпълнение за промяна на динамичното поведение на програмите

Многократно използване на имплементацията

- Наследяването не притежава тази гъвкавост, понеже компилаторът поставя ограничения по време на компилиране за класовете, създадени чрез наследяване
  - Понеже на наследяването се отделя голямо внимание, може да се помисли, че то може да се използва навсякъде
    - Понякога не е целесъобразно – може да доведе до трудни и прекалено сложни проекти
  - Вместо това, първо може да се разглежда композицията при създаване на нови класове, тъй като е по-просто и по-гъвкаво

Подходи за многократно използване

- Два подхода:
  - Композиция – използване обекти от съществуващи класове в новия клас
    - Използва се многократно функционалността на съществуващия код (не неговата форма)
  - Наследяване – новият клас се създава като тип на съществуващ клас
    - Използва се формата на съществуващ клас, като добавяме към нея код, без да променяме съществуващия клас
    - Компилаторът извършва по-голямата част от работата
    - Наследяването е базова концепция на ООП
- Синтаксисът и поведението на композицията и наследяването са сходни
  - Те представляват начини за създаване на нови типове от съществуващи

Синтаксис на композиция

- В много ОО системи се използва композиция
- Тя се реализира просто като поставим референции на обекти в рамките на нови класове
- Така, новите класове могат да съдържат:
  - Обекти от съществуващи класове – посредством референции
  - Примитиви – дефинират се директно

Пример

Пример

Какво прави програмата?

```
class WaterSource {
   private String s;
   WaterSource () {
      System.out.println("WaterSource ()");
      s = new String("Constructed");
    }
}
```

Пример

Какво прави програмата?

```
public class SprinklerSystem {
   private String valve1, valve2, valve3, valve4;
   WaterSource source;
   int i;
   float f;
   void print () {
      System.out.println("valve1 = " + valve1);
      System.out.println("valve2 = " + valve2);
      System.out.println("valve3 = " + valve3);
      System.out.println("valve4 = " + valve4);
      System.out.println("i = " + i);
      System.out.println("f = " + f);
      System.out.println("source = " + source);
   }
   public static void main(String[ ] args) {
      SprinklerSystem x = new SprinklerSystem ();
      x.print ();
   }
}
```

Пример

Резултат?

```
public class SprinklerSystem {
   private String valve1, valve2, valve3, valve4;
   WaterSource source;
   int i;
   float f;
   void print () {
      System.out.println("valve1 = " + valve1);
      System.out.println("valve2 = " + valve2);
      System.out.println("valve3 = " + valve3);
      System.out.println("valve4 = " + valve4);
      System.out.println("i = " + i);
      System.out.println("f = " + f);
      System.out.println("source = " + source);
   }
   public static void main(String[ ] args) {
      SprinklerSystem x = new SprinklerSystem ();
      x.print ();
   }
}
```

```
valve1 = null
valve2 = null
valve3 = null
valve4 = null
i = 0
f = 0.0
source = null

Process finished with exit code 0
```

Пример

Коментар?

```
public class SprinklerSystem {
   private String valve1, valve2, valve3, valve4;
   WaterSource source;
   int i;
   float f;
   void print () {
      System.out.println("valve1 = " + valve1);
      System.out.println("valve2 = " + valve2);
      System.out.println("valve3 = " + valve3);
      System.out.println("valve4 = " + valve4);
      System.out.println("i = " + i);
      System.out.println("f = " + f);
      System.out.println("source = " + source);
   }
   public static void main(String[ ] args) {
      SprinklerSystem x = new SprinklerSystem ();
      x.print ();
   }
}
```

```
valve1 = null
valve2 = null
valve3 = null
valve4 = null
i = 0
f = 0.0
source = null

Process finished with exit code 0
```

Компилаторът не извиква автоматично конструктора по подразбиране на WaterSource за да извърши инициализацията на source.

Инициализации

Инициализация на референции

- Примитивите, които са полета в клас се инициализират автоматично с 0.
- Референциите към обектите се инициализират с null - ако се опитваме да извикаме методи за някой от тях ще възникне изключение.
- Компилаторът не създава обект по подразбиране за референциите.

Инициализация на референции

- Инициализацията на референциите може да се направи явно по следните начини:
  - В момента на дефиниране на обектите - винаги ще бъдат инициализирани преди извикване на конструктора.
  - В конструктора на класа.
  - Непосредствено преди използване на обекта - съществуват ситуации, при които обектите не трябва да бъдат задължително създавани.

Пример

Какво прави програмата?

```
class Soap {
   private String s;
   Soap () {
      System.out.println("Soap ()");
      s = new String ("Constructed");
    }
   public String toString () { return s; }
}
```

Пример

Инициализация в момента на дефиниране

```
public class Bath {
     private String s1 = new String("Happy"), s2 = "Happy", s3, s4;
     Soap castille;
     int i; float toy;
     Bath () {
        System.out.println("Inside Bath ()");
        s3 = new String ("Joy");
        i = 47;
        toy = 3.14f;
        castille = new Soap ();
     }
     void print () {
      if (s4 == null) s4 = new String ("Joy");
      System.out.println ("s1 = " + s1);
      System.out.println ("s2 = " + s2);
      System.out.println ("s3 = " + s3);
      System.out.println ("s4 = " + s4);
      System.out.println ("i = " + i);
      System.out.println ("toy = " + toy);
      System.out.println ("castille = " + castille);
     }
    public static void main(String[ ] args) {
       Bath b = new Bath ();
       b.print ();
    }
}
```

Инициализация в конструктора

Късна инициализация

Пример

```
public class Bath {
     private String s1 = new String("Happy"), s2 = "Happy", s3, s4;
     Soap castille;
     int i; float toy;
    Bath () {
        System.out.println("Inside Bath ()");
        s3 = new String ("Joy");
        i = 47;
        toy = 3.14f;
        castille = new Soap ();
   }
  void print () {
      if (s4 == null) s4 = new String ("Joy");
      System.out.println ("s1 = " + s1);
      System.out.println ("s2 = " + s2);
      System.out.println ("s3 = " + s3);
      System.out.println ("s4 = " + s4);
      System.out.println ("i = " + i);
      System.out.println ("toy = " + toy);
      System.out.println ("castille = " + castille);
  }
 public static void main(String[ ] args) {
       Bath b = new Bath ();
       b.print ();
    }
}
```

```
Inside Bath ()
Soap ()
s1 = Happy
s2 = Happy
s3 = Joy
s4 = Joy
i = 47
toy = 3.14
castille = Constructed

Process finished with exit code 0
```

Пример

```
public class Bath {
     private String s1 = new String("Happy"), s2 = "Happy", s3, s4;
     Soap castille;
     int i; float toy;
     Bath () {
        System.out.println("Inside Bath ()");
        s3 = new String ("Joy");
        i = 47;
        toy = 3.14f;
        castille = new Soap ();
     }
     void print () {
        if (s4 == null) s4 = new String ("Joy");
        System.out.println ("s1 = " + s1);
        System.out.println ("s2 = " + s2);
        System.out.println ("s3 = " + s3);
        System.out.println ("s4 = " + s4);
        System.out.println ("i = " + i);
        System.out.println ("toy = " + toy);
        System.out.println  ("castille = " + castille);
     }
 public static void main(String[ ] args) {
       Bath b = new Bath ();
       b.print ();
    }
}
```

```
Inside Bath ()
Soap ()
s1 = Happy
s2 = Happy
s3 = Joy
s4 = Joy
i = 47
toy = 3.14
castille = Constructed

Proces finished with exit code 0
```

Това е референция към обект

Пример

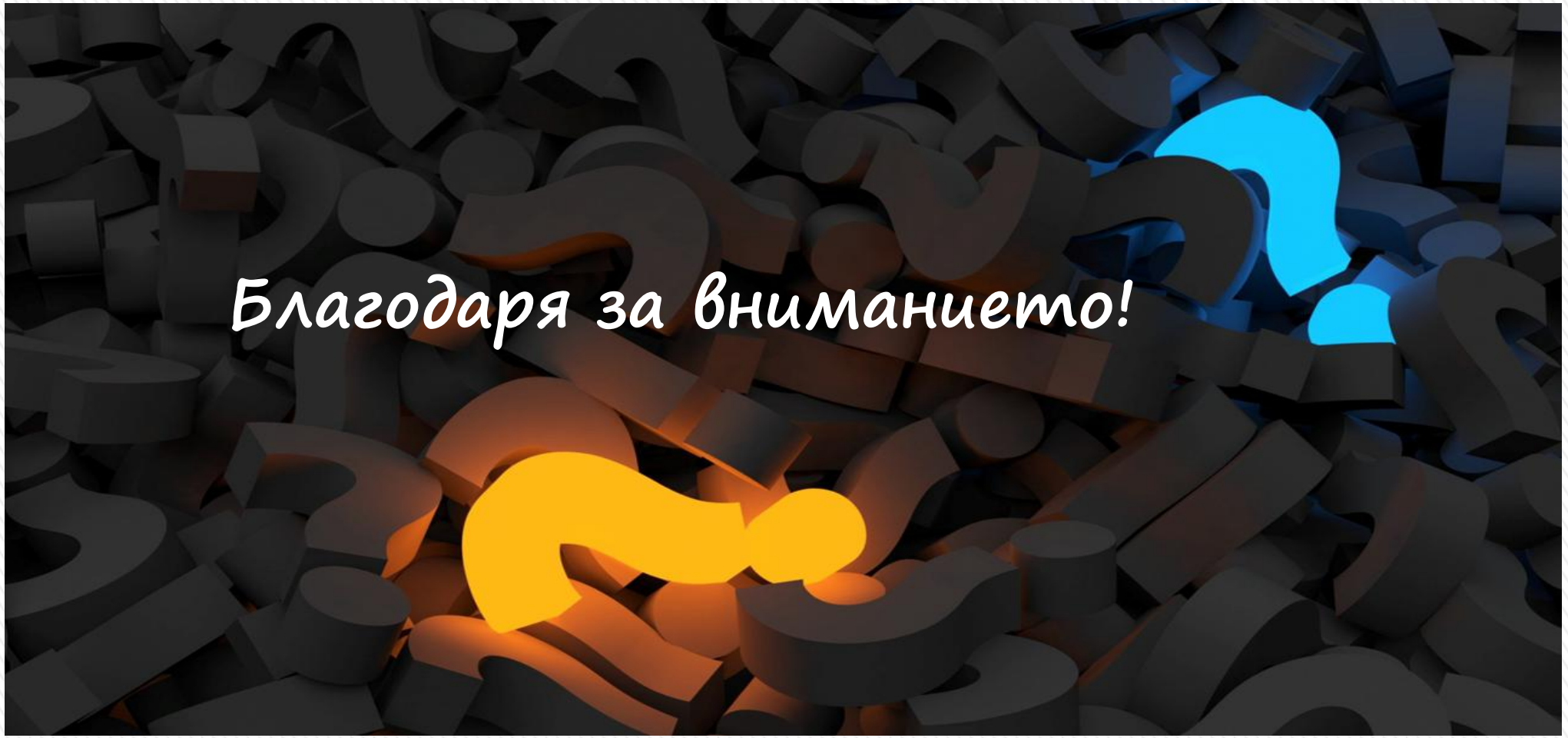
Какво прави програмата?

```
class Soap {
   private String s;
   Soap () {
      System.out.println("Soap ()");
      s = new String ("Constructed");
    }
   // public String toString () { return s;     }
}
```

```
Inside Bath ()
Soap ()
s1 = Happy
s2 = Happy
s3 = Joy
s4 = Joy
i = 47
toy = 3.14
castille = Soap@783e6358

Process finished with exit code 0
```

- Ако искаме да представим един обект като низ използваме метода toString() от класа Object.
- Методът toString() връща низовото представяне на обекта.


Благодаря за вниманието!